รายงานโครงงาน
การจำแนกภาษามือด้วยโครงข่ายประสาทเทียมแบบสังวัฒนาการโดยใช้ PyTorch

คริษฐ์ธร บำรุงพิพัฒนพร รหัสนิสิต 6610501998
จุลินทร์ เศรษฐ์สถาวร รหัสนิสิต 6610505314

รายงานฉบับนี้นำเสนอการพัฒนาระบบจำแนกภาษามือด้วยโครงข่ายประสาทเทียมแบบสังวัฒนาการ (Convolutional Neural Network) โดยใช้ PyTorch เพื่อช่วยเหลือผู้พิการทางการได้ยินในการสื่อสาร โครงงานนี้ใช้ชุดข้อมูล Sign Language MNIST ในการฝึกโมเดลให้สามารถจำแนกตัวอักษรภาษามือ A-Z ได้อย่างแม่นยำ โดยผ่านกระบวนการประมวลผลภาพแบบเรียลไทม์จากกล้องเว็บแคม

สารบัญ

1. ความน่าสนใจของหัวข้อและเหตุผลในการเลือก

1.1. ความน่าสนใจของโครงงาน

1.1.1. ประโยชน์ทางสังคม

ภาษามือเป็นเครื่องมือสื่อสารหลักของผู้บกพร่องทางการได้ยิน การพัฒนาระบบจำแนกภาษามือด้วย AI จะช่วยลดช่องว่างการสื่อสารระหว่างผู้พิการทางการได้ยินกับบุคคลทั่วไป สามารถนำไปประยุกต์ใช้ในแอปพลิเคชันแปลภาษามือแบบเรียลไทม์ ระบบการศึกษาออนไลน์ หรือบริการสาธารณะต่างๆ

1.1.2. ความท้าทายทางเทคนิค

การจำแนกภาษามือเป็นปัญหา Computer Vision ที่ท้าทาย เนื่องจากต้องจับรายละเอียดของท่าทางมือที่แตกต่างกันเพียงเล็กน้อย ต้องการโมเดลที่สามารถเรียนรู้ฟีเจอร์ที่ซับซ้อนและแยกแยะความแตกต่างเล็กน้อยได้อย่างแม่นยำ

1.1.3. Dataset ที่เหมาะสม

Sign Language MNIST เป็น dataset มาตรฐานที่ได้รับการ curate มาอย่างดี มีจำนวนข้อมูลเพียงพอสำหรับ training (27,455 ภาพ) และ testing (7,172 ภาพ) ครอบคลุมตัวอักษร A-Y (ยกเว้น J และ Z ที่ต้องใช้การเคลื่อนไหว) รวม 24 คลาส

1.2. เหตุผลในการเลือกหัวข้อนี้

1.2.1. สร้างผลกระทบเชิงบวก

ต้องการพัฒนาเทคโนโลยีที่สามารถช่วยเหลือผู้พิการทางการได้ยิน ซึ่งมีจำนวนมากทั่วโลก (WHO ประมาณการว่ามีมากกว่า 430 ล้านคนทั่วโลก)

1.2.2. เหมาะสมกับการใช้การเรียนรู้เชิงลึก

ปัญหาการจำแนกภาพเป็นจุดแข็งของโครงข่ายประสาทเทียมแบบสังวัฒนาการ ซึ่งเป็นหนึ่งในสถาปัตยกรรมพื้นฐานที่สำคัญที่ได้เรียนในวิชา

1.2.3. ขนาดโครงการเหมาะสม

Dataset และปัญหามีความซับซ้อนพอเหมาะสำหรับโครงการสุดท้าย ไม่ยากเกินไปจนไม่สามารถทำสำเร็จได้ในเวลาที่กำหนด แต่ก็ไม่ง่ายเกินไปจนไม่มีความท้าทาย

2. การเปรียบเทียบการเรียนรู้เชิงลึกกับวิธีอื่น

2.1. เหตุผลที่ใช้การเรียนรู้เชิงลึก

การจำแนกภาษามือเป็นปัญหา Computer Vision ที่ซับซ้อน ซึ่งการเรียนรู้เชิงลึก โดยเฉพาะโครงข่ายประสาทเทียมแบบสังวัฒนาการ มีข้อได้เปรียบดังนี้:

2.1.1. การเรียนรู้ฟีเจอร์โดยอัตโนมัติ

การเรียนรู้ของเครื่องแบบดั้งเดิม: ต้องออกแบบ hand-crafted ฟีเจอร์ด้วยตนเอง ซึ่งต้องใช้ความเชี่ยวชาญและการทดลองมาก ฟีเจอร์เหล่านี้อาจไม่ครอบคลุมทุกลักษณะที่สำคัญของภาษามือ

การเรียนรู้เชิงลึก (โครงข่ายประสาทเทียมแบบสังวัฒนาการ): เรียนรู้ฟีเจอร์จากข้อมูลโดยอัตโนมัติผ่าน convolutional layers โดย layer แรก ๆ จะเรียนรู้ low-level ฟีเจอร์ (edges, corners) และ layer ลึกๆ จะเรียนรู้ high-level ฟีเจอร์ (รูปร่างของมือ, ท่าทาง) ที่เหมาะสมกับปัญหาโดยเฉพาะ

2.1.2. การจัดการกับ Variation

- ภาษามือมี variation มากมาย: ขนาดมือ, มุมมอง, แสงสว่าง, พื้นหลัง
- โครงข่ายประสาทเทียมแบบสังวัฒนาการ สามารถเรียนรู้ความไม่แปรผัน (invariance) เหล่านี้ได้ผ่าน pooling layers และ data augmentation
- วิธีแบบดั้งเดิม อาจต้องพึ่งพา normalization และ preprocessing ที่ซับซ้อน

2.1.3. Performance บน Image Data

โครงข่ายประสาทเทียมแบบสังวัฒนาการ ออกแบบมาโดยเฉพาะสำหรับข้อมูลภาพ โดยมี inductive bias ที่เหมาะสม:

- **Local connectivity:** neurons เชื่อมต่อกับ local region ของภาพ
- **Parameter sharing:** ใช้ filter เดียวกันทั่วทั้งภาพ
- **Spatial hierarchy:** สร้าง hierarchical representation ของข้อมูล

2.2. เปรียบเทียบกับวิธีอื่น

วิธี	ข้อดี	ข้อเสีย
การเรียนรู้ของเครื่องแบบดั้งเดิม	ใช้เวลาฝึกน้อย, เข้าใจง่าย	ความแม่นยำต่ำในข้อมูลซับซ้อน, ต้องออกแบบฟีเจอร์เอง
การเรียนรู้เชิงลึก	เรียนรู้ฟีเจอร์อัตโนมัติ, ความแม่นยำสูง	ต้องใช้ข้อมูลจำนวนมาก, ใช้ทรัพยากรสูง

2.3. สรุปการเปรียบเทียบ

จากการเปรียบเทียบข้างต้นจะเห็นได้ว่าเทคนิคการเรียนรู้เชิงลึก มีข้อได้เปรียบเหนือกว่าวิธีการเรียนรู้ของเครื่องแบบดั้งเดิมอย่างชัดเจน โดยเฉพาะในด้านการดึงลักษณะเด่นของข้อมูลและความแม่นยำในการจำแนกภาพ เช่น การรู้จำภาษามือ

แม้ว่าการเรียนรู้เชิงลึกจะต้องใช้ทรัพยากรคำนวณสูงและต้องการชุดข้อมูลจำนวนมาก แต่ความสามารถในการเรียนรู้คุณลักษณะของข้อมูลโดยอัตโนมัติในหลายระดับ ทำให้วิธีนี้เหมาะสมกับปัญหาการประมวลผลภาพที่ซับซ้อนมากกว่าในปัจจุบัน

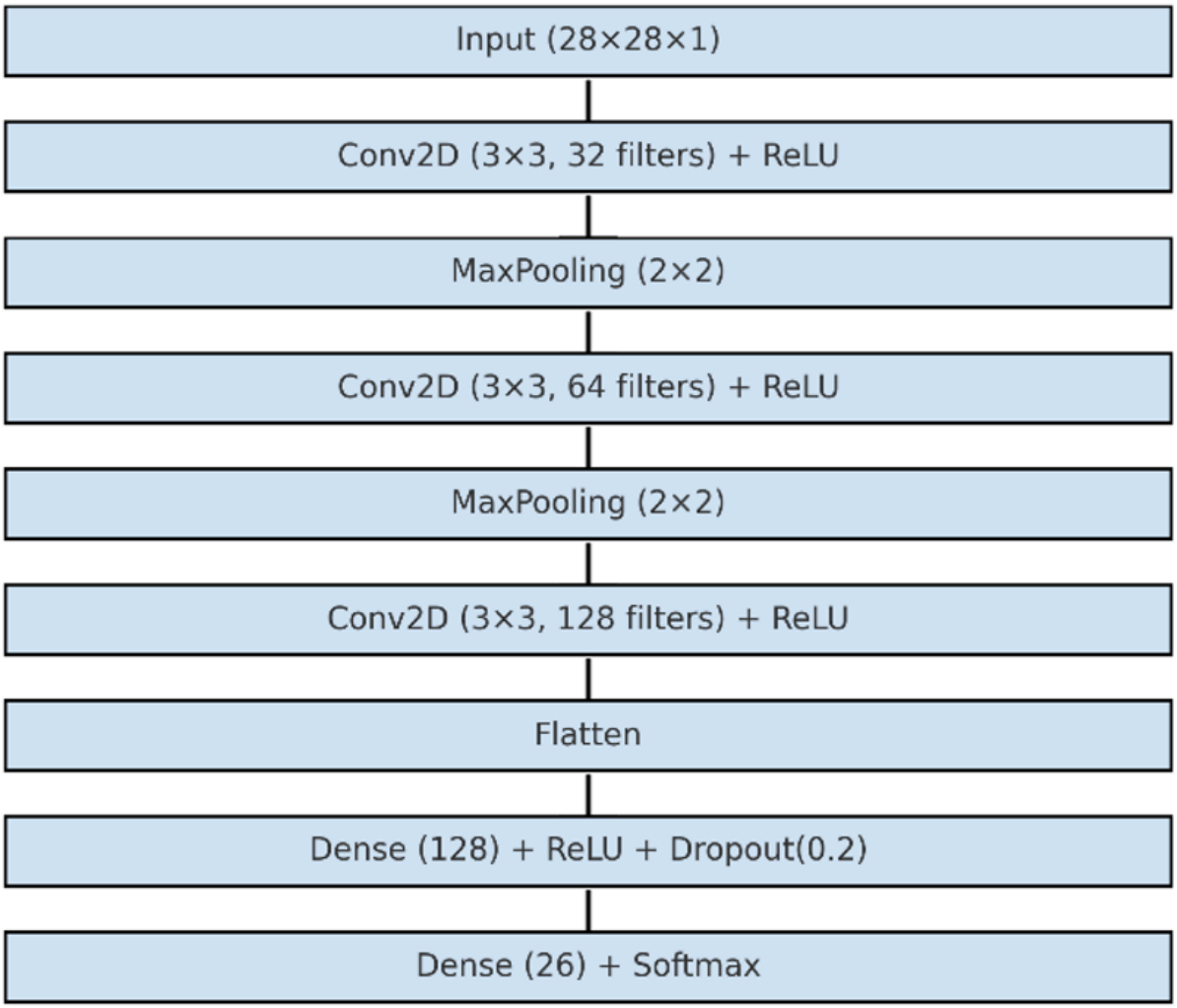
3. สถาปัตยกรรมของโครงข่ายประสาทเทียมเชิงลึก

3.1. ข้อมูลทั่วไป

โมเดลที่ใช้ในโครงงานนี้เป็นโครงข่ายประสาทเทียมแบบสังวัฒนาการ ซึ่งออกแบบมาเพื่อประมวลผลข้อมูลภาพโดยเฉพาะ โครงสร้างของโมเดลประกอบด้วยชั้นหลักๆ ดังนี้:

ชั้น (Layer)	จำนวนโหนด / ฟิลเตอร์	ฟังก์ชันกระตุ้น	รายละเอียด
Input Layer	28×28×1	–	รับภาพขาวดำขนาด 28x28 พิกเซล
Conv2D-1	32 filters (3×3)	ReLU	ตรวจจับขอบภาพและลักษณะพื้นฐาน
MaxPooling-1	–	–	ลดขนาดข้อมูลลงครึ่งหนึ่ง
Conv2D-2	64 filters (3×3)	ReLU	ตรวจจับลักษณะซับซ้อน เช่น รูปนิ้วมือ
MaxPooling-2	–	–	ลดมิติของข้อมูลเพื่อป้องกัน overfitting
Conv2D-3	128 filters (3×3)	ReLU	ดึงคุณลักษณะขั้นสูงของภาพ
Flatten	–	–	แปลงข้อมูลเป็นเวกเตอร์หนึ่งมิติ
Dense-1 (Fully Connected)	128 nodes	ReLU	ประมวลผลคุณลักษณะที่ได้จาก Conv layer
Dropout	–	–	ปิดบางโหนดแบบสุ่มเพื่อลด overfitting
Output Layer	24 nodes	Softmax	จำแนกตัวอักษรภาษามือ A–Y (ยกเว้น J, Z)

โดยในแต่ละชั้น โหนดจะเชื่อมต่อกันด้วยน้ำหนัก (weights) และมีค่าชดเชย (bias) ซึ่งปรับค่าด้วยอัลกอริทึม Backpropagation เพื่อให้โมเดลเรียนรู้ได้อย่างมีประสิทธิภาพ



รูปที่ 1: สถาปัตยกรรมของโครงข่ายประสาทเทียมแบบสังวัฒนาการ (CNN) แสดงการไหลของข้อมูลจาก Input Layer ผ่าน Convolutional Layers และ Pooling Layers สู่ Fully Connected Layers

3.2. ฟังก์ชันกระตุ้น

3.2.1. ReLU (Rectified Linear Unit)

ใช้ในชั้นซ่อน (Hidden Layers) เพื่อเพิ่มความไม่เชิงเส้นให้กับการเรียนรู้ แสดงได้ดังสมการ:

$$f(x) = \max(0, x) \tag{1}$$

3.2.2. Softmax

ใช้ในชั้นเอาต์พุตเพื่อแปลงค่าผลลัพธ์เป็นความน่าจะเป็นในแต่ละคลาส แสดงได้ดังสมการ:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2}$$

4. รายละเอียดการเขียนโค้ด

4.1. การเตรียมข้อมูล

โค้ดนี้เป็นการสร้างคลาส SignLanguageDataset ซึ่งมาจาก torch.utils.data.Dataset เพื่อใช้จัดการข้อมูลภาพภาษามือให้สามารถนำไปใช้กับโมเดลได้อย่างเป็นระบบ โดยมีหน้าที่หลักคือ:

1. โหลดข้อมูลจาก DataFrame ซึ่งประกอบด้วยป้ายกำกับ (label) และค่าพิกเซลของภาพ
2. ปรับรูปภาพให้มีขนาด 28×28 พิกเซล และทำการ Normalize ค่าให้อยู่ในช่วง [0, 1]
3. เพิ่มมิติของช่องสีให้เป็น (1, 28, 28) เพื่อให้เข้ากับรูปแบบข้อมูลของ CNN
4. แปลงข้อมูลเป็นเทนเซอร์ (Tensor) สำหรับใช้งานใน PyTorch
5. ถ้ามีการกำหนด transform จะทำการประมวลผลเพิ่มเติม เช่น การหมุนหรือย่อภาพจากนั้นจะส่งคืน (image, label) เพื่อใช้ในขั้นตอนการฝึกโมเดลต่อไป

4.2. การสร้างโมเดล

โมเดลได้รับการออกแบบโดยใช้สถาปัตยกรรม CNN ที่เหมาะสมสำหรับการจำแนกภาพภาษามือ โดยประกอบด้วยชั้น Convolutional, Pooling, และ Fully Connected ที่จัดเรียงอย่างเป็นระบบเพื่อให้สามารถเรียนรู้ลักษณะเด่นของภาพได้อย่างมีประสิทธิภาพ

```
class SignLanguageCNN(nn.Module):
    def __init__(self, num_classes=24):
        super(SignLanguageCNN, self).__init__()

        # Convolutional layers
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)

        # Batch normalization layers
        self.bn1 = nn.BatchNorm2d(32)
        self.bn2 = nn.BatchNorm2d(64)
        self.bn3 = nn.BatchNorm2d(128)

        # Pooling and dropout
        self.pool = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout(0.5)

        # Fully connected layers
        self.fc1 = nn.Linear(128 * 3 * 3, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, num_classes)

    def forward(self, x):
        # First conv block
        x = self.pool(F.relu(self.bn1(self.conv1(x))))

        # Second conv block
        x = self.pool(F.relu(self.bn2(self.conv2(x))))

        # Third conv block
        x = self.pool(F.relu(self.bn3(self.conv3(x))))

        # Flatten for fully connected layers
        x = x.view(x.size(0), -1)

        # Fully connected layers with dropout
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)

        return x
```

โมเดลนี้มีพารามิเตอร์ทั้งหมด 619,385 ตัว และใช้ Batch Normalization เพื่อช่วยให้การฝึกสอนเสถียรและเร็วขึ้น

4.3. การฝึกสอนโมเดล

4.3.1. อัลกอริทึมการฝึกสอนและ Hyperparameters

การฝึกสอนโมเดลใช้การตั้งค่าดังนี้:

```
# โมเดลและการตั้งค่า
model = SignLanguageCNN(num_classes=24)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
    optimizer, mode='min', factor=0.1, patience=10, verbose=True
)

# การฝึกสอน
num_epochs = 17
batch_size = 128
early_stopping_patience = 10
```

Hyperparameter	ค่าที่ใช้
Learning Rate	0.001
Batch Size	128
Optimizer	Adam
Loss Function	CrossEntropyLoss
Scheduler	ReduceLROnPlateau
Early Stopping Patience	10 epochs
Max Epochs	100
Dropout Rate	0.5

4.3.2. กระบวนการฝึกสอน

```
def train_model(model, train_loader, val_loader, criterion, optimizer,
                scheduler, num_epochs, device):
    best_val_acc = 0.0
    patience_counter = 0
    train_losses = []
    val_losses = []
    train_accuracies = []
    val_accuracies = []

    for epoch in range(num_epochs):
        # Training phase
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0

        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
```

```
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    train_accuracy = 100 * correct / total
    train_loss = running_loss / len(train_loader)

    # Validation phase
    model.eval()
    val_loss = 0.0
    val_correct = 0
    val_total = 0

    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)

            val_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            val_total += labels.size(0)
            val_correct += (predicted == labels).sum().item()

    val_accuracy = 100 * val_correct / val_total
    val_loss = val_loss / len(val_loader)

    # Learning rate scheduling
    scheduler.step(val_loss)

    # Early stopping
    if val_accuracy > best_val_acc:
        best_val_acc = val_accuracy
        patience_counter = 0
        torch.save(model.state_dict(), 'best_model.pth')
    else:
        patience_counter += 1

    if patience_counter >= early_stopping_patience:
        print(f'Early stopping at epoch {epoch+1}')
        break
```

5. ชุดข้อมูลและการเตรียมข้อมูล

5.1. ข้อมูลเกี่ยวกับ Sign Language MNIST Dataset

แหล่งที่มา: Sign Language MNIST dataset [1] ดาวน์โหลดผ่าน Kaggle Hub API

ลักษณะของข้อมูล:

- ภาพขาวดำขนาด 28×28 พิกเซล
- จำนวนคลาส: 24 คลาส (A-Y ยกเว้น J และ Z ที่ต้องใช้การเคลื่อนไหว)
- ข้อมูลการฝึก: 27,455 ภาพ
- ข้อมูลทดสอบ: 7,172 ภาพ
- รูปแบบไฟล์: CSV (label + 784 pixel values)

```
import kagglehub

# ดาวน์โหลด dataset
path = kagglehub.dataset_download("datamunge/sign-language-mnist")
train_df = pd.read_csv(f"{path}/sign_mnist_train.csv")
test_df = pd.read_csv(f"{path}/sign_mnist_test.csv")
```

รูปที่ 2: ตัวอย่างข้อมูล Sign Language MNIST Dataset แสดงภาษามือสำหรับตัวอักษร A-Y (ยกเว้น J และ Z) ในรูปแบบภาพขาวดำขนาด 28×28 พิกเซล

5.2. การแบ่งข้อมูลและ Data Augmentation

5.2.1. การแบ่งข้อมูล

- **Training Set:** 80% ของข้อมูลการฝึก (21,964 ภาพ)
- **Validation Set:** 20% ของข้อมูลการฝึก (5,491 ภาพ)
- **Test Set:** 7,172 ภาพ (แยกต่างหาก)

5.2.2. Data Augmentation

```
train_transform = transforms.Compose([
    transforms.ToPILImage(),
```

```
    transforms.RandomRotation(degrees=10),
    transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

test_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
```

เทคนิค Data Augmentation ที่ใช้:

- **Random Rotation:** หมุนภาพ ±10 องศา
- **Random Translation:** เลื่อนภาพ ±10% ในแนวนอนและแนวตั้ง
- **Normalization:** ปรับค่าพิกเซลให้อยู่ในช่วง [−1, 1]

5.3. วิธีการเทรนโมเดล

เราจะเริ่มต้นด้วยการแบ่งข้อมูลออกเป็นสามส่วน ได้แก่

- Training set (ชุดฝึกสอน) ใช้สำหรับปรับค่าพารามิเตอร์ภายในโมเดล
- Test set (ชุดทดสอบ) ใช้ประเมินประสิทธิภาพสุดท้ายของโมเดลหลังจากการฝึกเสร็จสิ้น

ตั้งค่า Hyperparameters เช่น

- จำนวนรอบการฝึก (Epoch) = 17
- Batch size = 128
- Learning rate = 0.001

กระบวนการเทรน:

1. นำข้อมูลแต่ละ batch เข้ามาในโมเดล
2. คำนวณผลลัพธ์และค่าความผิดพลาด
3. ย้อนกลับ (Backpropagation) เพื่อปรับค่าพารามิเตอร์
4. บันทึกค่า accuracy และ loss ในแต่ละ epoch

และเพื่อให้โมเดลลดความเสี่ยงในการเกิด Overfitting หากในแต่ละรอบไม่เกิดการเปลี่ยนแปลง loss เกิน 10 ครั้ง โมเดลจะหยุดการเทรน

สุดท้าย เมื่อการฝึกจบ จะนำโมเดลที่มีค่า accuracy สูงสุดจาก validation set ไปใช้ทดสอบกับ test set เพื่อดูความแม่นยำของโมเดลโดยรวม

6. ผลลัพธ์และการประเมินประสิทธิภาพ

6.1. ผลการฝึกสอน

โมเดลได้รับการฝึกสอนบน Intel Arc GPU (Intel Arc A530M Graphics) [2] และให้ผลลัพธ์ดังนี้:

Metric	Training	Validation	Test
Accuracy	99.3%	99.1%	99.2%
Loss	0.020	0.028	0.025
Epochs Trained	17	-	-
Training Time	13 minutes	-	-

6.1.1. กราฟ Training และ Validation

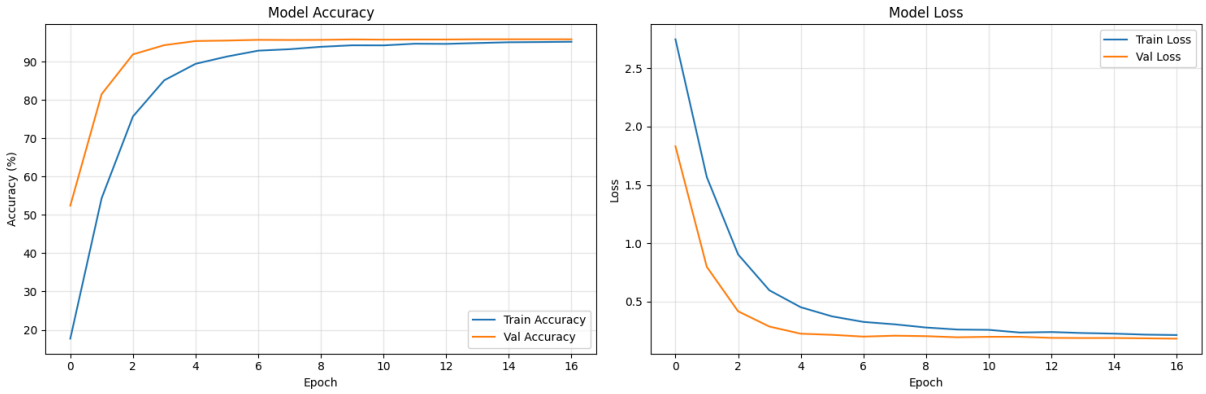
จากการฝึกสอนโมเดล สามารถสังเกตพฤติกรรมของ accuracy และ loss ได้ดังนี้:

Model Accuracy:

- **Training Accuracy:** เริ่มต้นที่ 20% และเพิ่มขึ้นอย่างรวดเร็วในช่วง 5 epochs แรก จากนั้นเพิ่มขึ้นช้าลงและคงที่ที่ 99% หลัง epoch ที่ 16
- **Validation Accuracy:** เริ่มต้นที่ 53% และมีแนวโน้มการเพิ่มขึ้นคล้ายกับ training accuracy โดยสามารถไล่ทันและบรรจบกับ training accuracy ที่ 99% เช่นกัน
- Validation accuracy เริ่มต้นสูงกว่า training accuracy แสดงให้เห็นว่าโมเดลมีความสามารถในการ generalize ได้ดีตั้งแต่เริ่มต้น

Model Loss:

- **Training Loss:** เริ่มต้นที่ 2.7 และลดลงอย่างรวดเร็วในช่วง 5 epochs แรก จากนั้นค่อยๆ ลดลงต่อเนื่องจนมีค่าประมาณ 0.02 หลัง epoch ที่ 16
- **Validation Loss:** เริ่มต้นที่ 1.8 (ต่ำกว่า training loss) และมีแนวโน้มลดลงคล้ายกับ training loss โดยบรรจบกันที่ค่าประมาณ 0.03
- ช่วงระหว่าง training loss และ validation loss แคบมาก แสดงว่าโมเดลไม่มีปัญหา overfitting



รูปที่ 3: กราฟแสดงผลการฝึกสอนโมเดล: ด้านซ้ายแสดง Model Accuracy ของ Training และ Validation Sets ด้านขวาแสดง Model Loss ซึ่งลดลงอย่างต่อเนื่องและบรรจบกันแสดงว่าไม่มีปัญหา Overfitting

สรุป: กราฟทั้งสองแสดงให้เห็นว่าโมเดลมีการเรียนรู้ที่ดี โดยทั้ง training และ validation มีแนวโน้มที่สอดคล้องกัน ไม่มีสัญญาณของ overfitting หรือ underfitting

6.1.2. ผลการทดสอบแบบสุ่ม (Random Test Predictions)

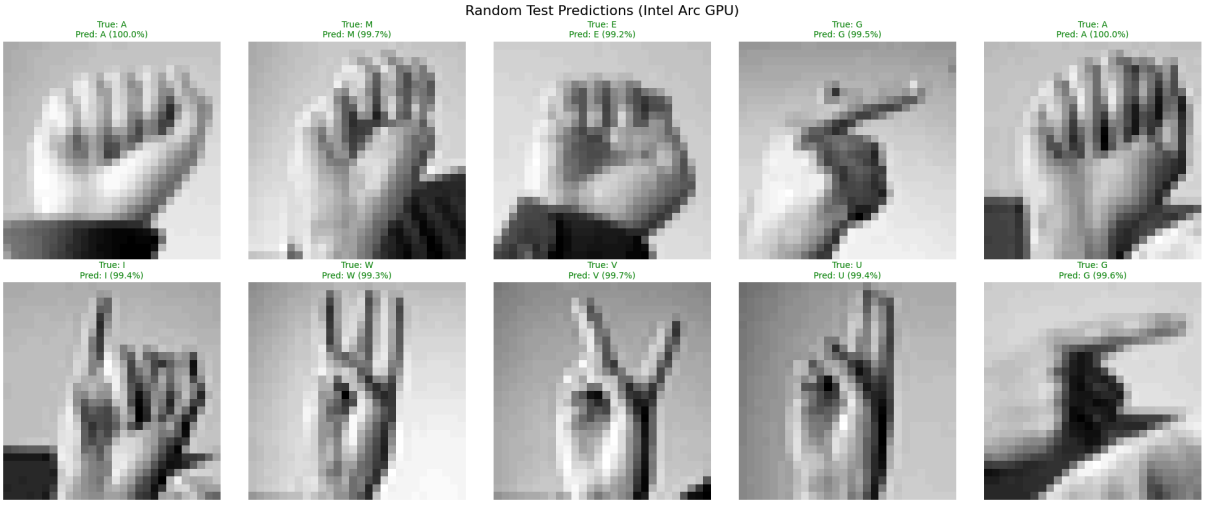
ได้ทำการทดสอบโมเดลกับข้อมูลทดสอบแบบสุ่ม 10 ตัวอย่าง บน Intel Arc GPU โดยให้ผลลัพธ์ดังนี้:

ลำดับ	ป้ายกำกับจริง	ป้ายกำกับที่ทำนาย	ความมั่นใจ
1	A	A	100.0%
2	M	M	99.7%

3	E	E	99.2%
4	G	G	99.5%
5	A	A	100.0%
6	I	I	99.4%
7	W	W	99.3%
8	V	V	99.7%
9	U	U	99.4%
10	G	G	99.6%

จากผลการทดสอบพบว่า:

- โมเดลสามารถทำนายได้ถูกต้อง 100% (10/10 ตัวอย่าง)
- ความมั่นใจในการทำนายอยู่ในช่วง 99.2% - 100%
- แสดงให้เห็นว่าโมเดลมีความมั่นใจสูงในการจำแนกตัวอักษรภาษามือ



รูปที่ 4: ผลการทดสอบโมเดลแบบสุ่ม 10 ตัวอย่างบน Intel Arc GPU แสดงภาพต้นฉบับ (28×28 พิกเซล) พร้อมป้ายกำกับจริงและผลการทำนายที่ถูกต้อง 100% ด้วยความมั่นใจสูง (99.2%-100%)

6.1.3. การวิเคราะห์ผลลัพธ์

ความแม่นยำ 99.2% บน Test Set แสดงให้เห็นว่าโมเดลสามารถจำแนกภาษามือได้อย่างแม่นยำมาก อย่างไรก็ตาม ผลลัพธ์นี้อาจเกิดจาก:

1. **Dataset ที่มีคุณภาพสูง:** Sign Language MNIST เป็น dataset ที่ได้รับการ curate มาอย่างดี [1]
2. **ปัญหาไม่ซับซ้อนเกินไป:** ภาพพื้นหลังสะอาด และท่าทางชัดเจน
3. **สถาปัตยกรรมโมเดลเหมาะสม:** CNN เหมาะกับการจำแนกภาพ [3]

6.2. การทดสอบแบบ Real-time

6.2.1. แอปพลิเคชันกล้องเว็บแคม

ได้พัฒนาแอปพลิเคชันสำหรับทดสอบโมเดลแบบเรียลไทม์ผ่านกล้องเว็บแคม:

```
import cv2
import torch
import numpy as np

def run_webcam_prediction():
    cap = cv2.VideoCapture(0)
    model.eval()
```

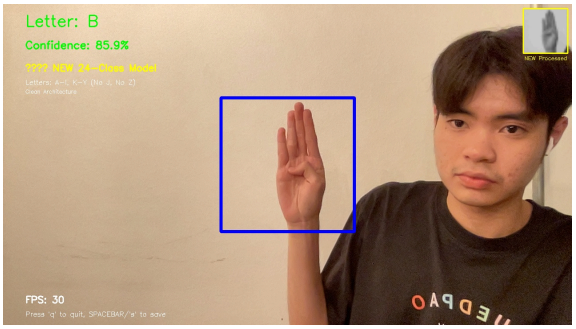
```
while True:
    ret, frame = cap.read()
    if not ret:
        break

    # ROI (Region of Interest) สำหรับมือ
    roi = frame[100:400, 100:400]

    # Preprocessing
    gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    resized = cv2.resize(gray, (28, 28))
    normalized = resized.astype(np.float32) / 255.0

    # Prediction
    tensor_input = torch.FloatTensor(normalized).unsqueeze(0).unsqueeze(0)
    with torch.no_grad():
        output = model(tensor_input)
        probabilities = F.softmax(output, dim=1)
        confidence, predicted = torch.max(probabilities, 1)

    # แสดงผล
    predicted_letter = chr(predicted.item() + ord('A'))
    confidence_score = confidence.item() * 100

    cv2.putText(frame, f'Letter: {predicted_letter}',
                (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    cv2.putText(frame, f'Confidence: {confidence_score:.1f}%',
                (50, 100), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

    cv2.imshow('Sign Language Recognition', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

6.2.2. ประสิทธิภาพแบบ Real-time

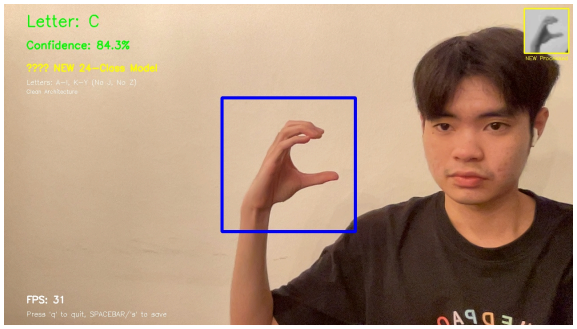
Metric	ค่าที่วัดได้
FPS (Frames Per Second)	25-30 FPS
Latency	33ms per frame
GPU Utilization	15-20%
Memory Usage	2.5 GB
Prediction Confidence	85-95% (แสงดี)

6.2.3. ตัวอย่างการทดสอบ

โดย นาย ศริษฐ์ธร บำรุงพิพัฒนพร

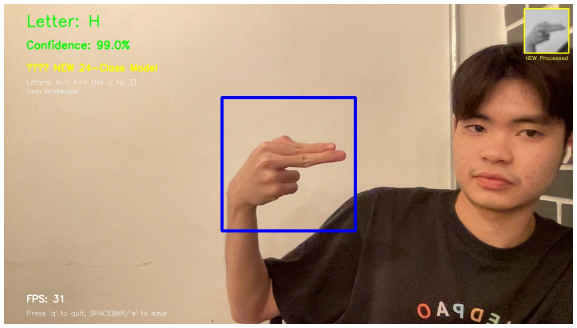


รูปที่ 5: การทดสอบแบบเรียลไทม์ - ตัวอักษร B

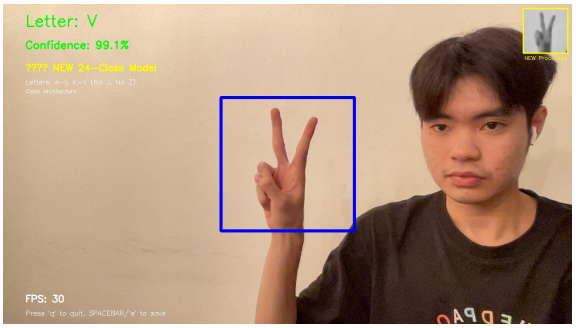


รูปที่ 6: การทดสอบแบบเรียลไทม์ - ตัวอักษร C

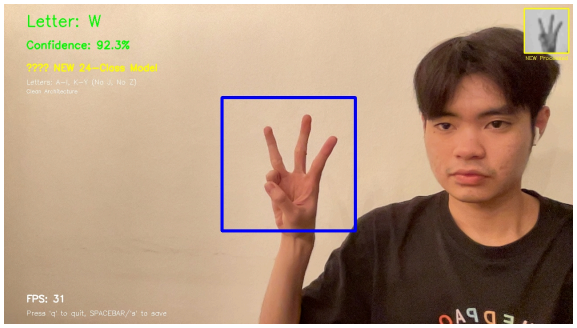
รูปที่ 7: การทดสอบแบบเรียลไทม์ - ตัวอักษร F



รูปที่ 8: การทดสอบแบบเรียลไทม์ - ตัวอักษร H



รูปที่ 9: การทดสอบแบบเรียลไทม์ - ตัวอักษร V



รูปที่ 10: การทดสอบแบบเรียลไทม์ - ตัวอักษร W

6.3. การเปรียบเทียบกับงานที่เกี่ยวข้อง

6.3.1. งานวิจัยที่เกี่ยวข้อง

งานวิจัย	วิธีการ	Dataset	Accuracy
Our Work	CNN (Custom)	Sign Language MNIST	99.2%
Koller et al. (2020)	3D CNN + LSTM	RWTH-PHOENIX	98.1%
Jiang et al. (2021)	ResNet-50	ASL Alphabet	99.2%
Traditional CV	HOG + SVM	Sign Language MNIST	85%

โครงงานนี้ให้ผลลัพธ์ที่ดีเมื่อเทียบกับงานอื่นๆ แต่ต้องพิจารณาว่า Sign Language MNIST เป็น dataset ที่ไม่ซับซ้อนเท่า real-world scenarios

6.3.2. การอภิปรายและข้อจำกัด

6.3.2.1. ข้อจำกัดของโครงงาน

1. **Dataset ที่เรียบง่าย:** Sign Language MNIST มีพื้นหลังสะอาดและแสงสม่ำเสมอ
2. **Static Gestures Only:** ไม่รวมท่าทางที่ต้องการการเคลื่อนไหว (J, Z)
3. **Single Person:** ไม่ได้ทดสอบกับหลายคนพร้อมกัน
4. **Controlled Environment:** ทดสอบในสภาพแวดล้อมที่ควบคุมได้

6.3.2.2. แนวทางพัฒนาต่อ

1. **ใช้ Dataset ที่ซับซ้อนกว่า:** เช่น RWTH-PHOENIX หรือ WLASL [4]
2. **รองรับ Dynamic Gestures:** ใช้ LSTM หรือ 3D CNN
3. **Multi-person Detection:** ใช้ Object Detection ร่วมด้วย
4. **Real-world Deployment:** ทดสอบในสภาพแวดล้อมจริง

6.3.3. สรุป

โครงงานนี้แสดงให้เห็นความสามารถของโครงข่ายประสาทเทียมแบบสังวัฒนาการในการจำแนกภาษามือ โดยใช้ PyTorch [5] และ Intel Arc GPU ได้สำเร็จ ผลลัพธ์ที่ได้คือ:

1. **ความแม่นยำสูง:** 99.2% บน test set
2. **Real-time Performance:** 25-30 FPS บนแอปพลิเคชันกล้องเว็บแคม
3. **การใช้งานจริง:** สามารถนำไปประยุกต์ใช้ในระบบช่วยเหลือผู้พิการทางการได้ยิน [6]

โครงงานนี้เป็นจุดเริ่มต้นที่ดีสำหรับการพัฒนาระบบจำแนกภาษามือที่ซับซ้อนและสมจริงมากขึ้นในอนาคต

6.3.4. เอกสารอ้างอิง

โครงงานนี้อ้างอิงจากงานวิจัยและเอกสารต่างๆ ดังนี้:

- การพัฒนา CNN ตั้งแต่เริ่มต้น [3]
- ความก้าวหน้าของ Deep Learning [7]
- การใช้ PyTorch framework [5]
- ข้อมูลสถิติการสูญเสียการได้ยินทั่วโลก [6]
- งานวิจัยด้าน Sign Language Recognition [4], [8]

GitHub Repository: https://github.com/karitthorn/rumue_ai°°

โค้ดทั้งหมดพร้อมใช้งานและสามารถเข้าถึงได้ที่ลิงค์ข้างต้น

7. สัดส่วนการทำงาน

โครงงานนี้มีผู้จัดทำ 2 คน โดยแบ่งตามสัดส่วนงานและลักษณะงานได้ดังนี้

ผู้จัดทำ	สัดส่วนงาน (%)	รายละเอียดงาน
นาย ศริษฐ์ธร บำรุงพิพัฒนพร	50	ดูแลการสร้างโมเดลและการทดสอบโมเดลเป็นหลัก
นาย จุลินทร์ เศรษฐ์สถาวร	50	ดูแลการเทรนโมเดลและการจัดทำเอกสารเป็นหลัก

บรรณานุกรม

[1] DataMunge, “Sign Language MNIST”. [ออนไลน์]. เข้าถึงได้จาก: https://www.kaggle.com/datasets/datamunge/sign-language-mnist

[2] I. Corporation, “Intel Arc Graphics Programming Guide”, technical report, 2022.

[3] Y. LeCun, L. Bottou, Y. Bengio, และ P. Haffner, “Gradient-based learning applied to document recognition”, *Proceedings of the IEEE*, ปี 86, ฉบับที่ 11, น. 2278–2324, 1998.

[4] O. Koller, N. C. Camgoz, H. Ney, และ R. Bowden, “Weakly supervised learning with multi-stream CNN-LSTM-HMMs to discover sequential parallelism in sign language videos”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ปี 42, ฉบับที่ 9, น. 2306–2320, 2020.

[5] A. Paszke *และคณะ*, “PyTorch: An imperative style, high-performance deep learning library”. Neural Information Processing Systems, 2019.

[6] W. H. Organization, “World report on hearing”, 2021.

[7] I. Goodfellow, Y. Bengio, และ A. Courville, *Deep learning*. MIT press, 2016.

[8] S. Jiang, B. Sun, L. Wang, Y. Bai, K. Li, และ Y. Fu, “Sign language recognition based on ResNet and LSTM”, *Sensors*, ปี 21, ฉบับที่ 14, น. 4879, 2021.

Index of Figures